

PROJET-JAVA

Java project by Peter and Younes

Younes was charged with the design of the graphical user interface and Peter handled the synchronization logic in local and network and we both worked on the UML.

Project which consists in synchronizing two folders in Java, whether it's in local or in network, without the help of extern API.

The synchronization is possible on a single computer in local, or on network in local, using 2 computers. The synchronization works both ways: if the folder A is modified, then the folder B is also modified, and the way around.

To send data through sockets, we first serialize the data in a String buffer, then the receiver deserialize the data, using the separators we have chosen "||". The receiver receives in order the type of the file : 1 if it's a folder, 0 if it's a file, then the relative path of the file, and finally the content of the file, if it's one.

There is a lot of recursion in the project, as we can't directly send a folder with its file, or copy a whole folder. To do so, we first create the folder, then call the method recursively, to ensure that every subfolders are sent.

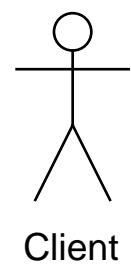
Execution : `java -jar peter_younes.jar` or open the jar file.

To make it work in local, on a single computer, the user must choose the source folder he wants to synchronize, and the destination folder, where it needs to be synchronized, then press the Synchronize button.

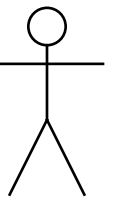
To make it work in network, the server must choose a folder where he will receive the files, and choose a port, then launch the server. Only then, the client can choose the folder he wishes to send, choose the same port as the server, and connect to it.

The server must be started first, before the client can connect to it, otherwise there will be an error.

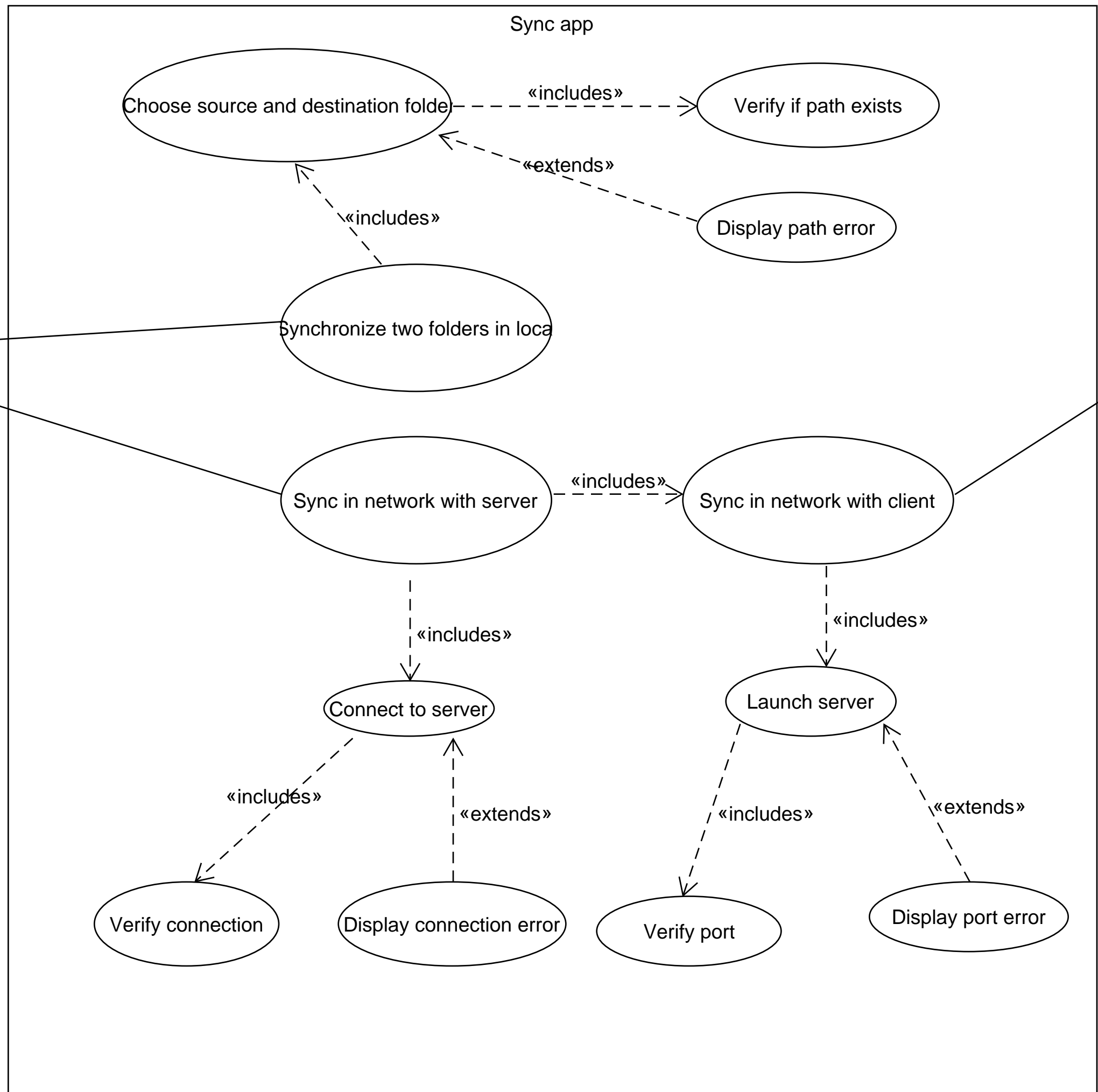
Use case



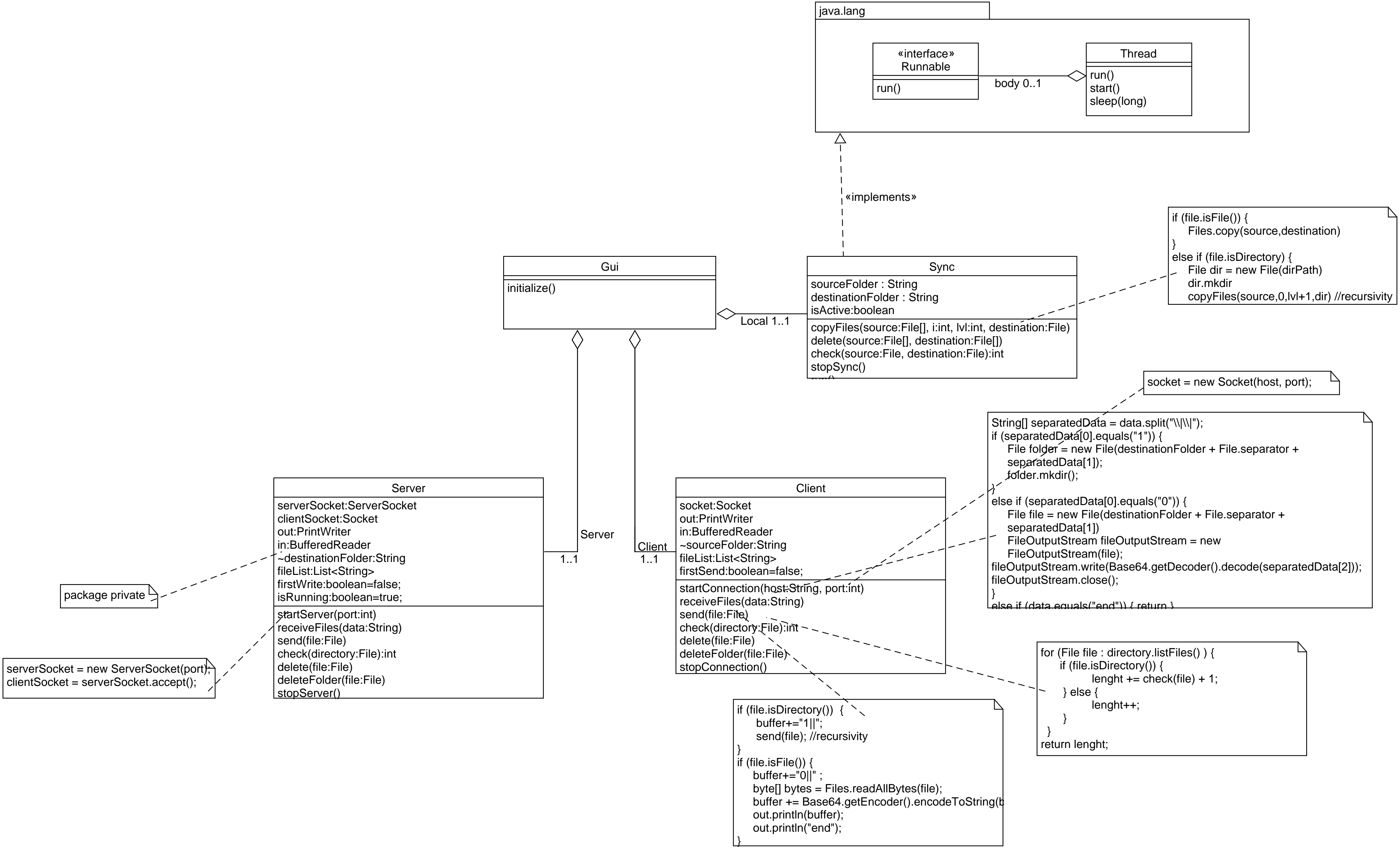
Client



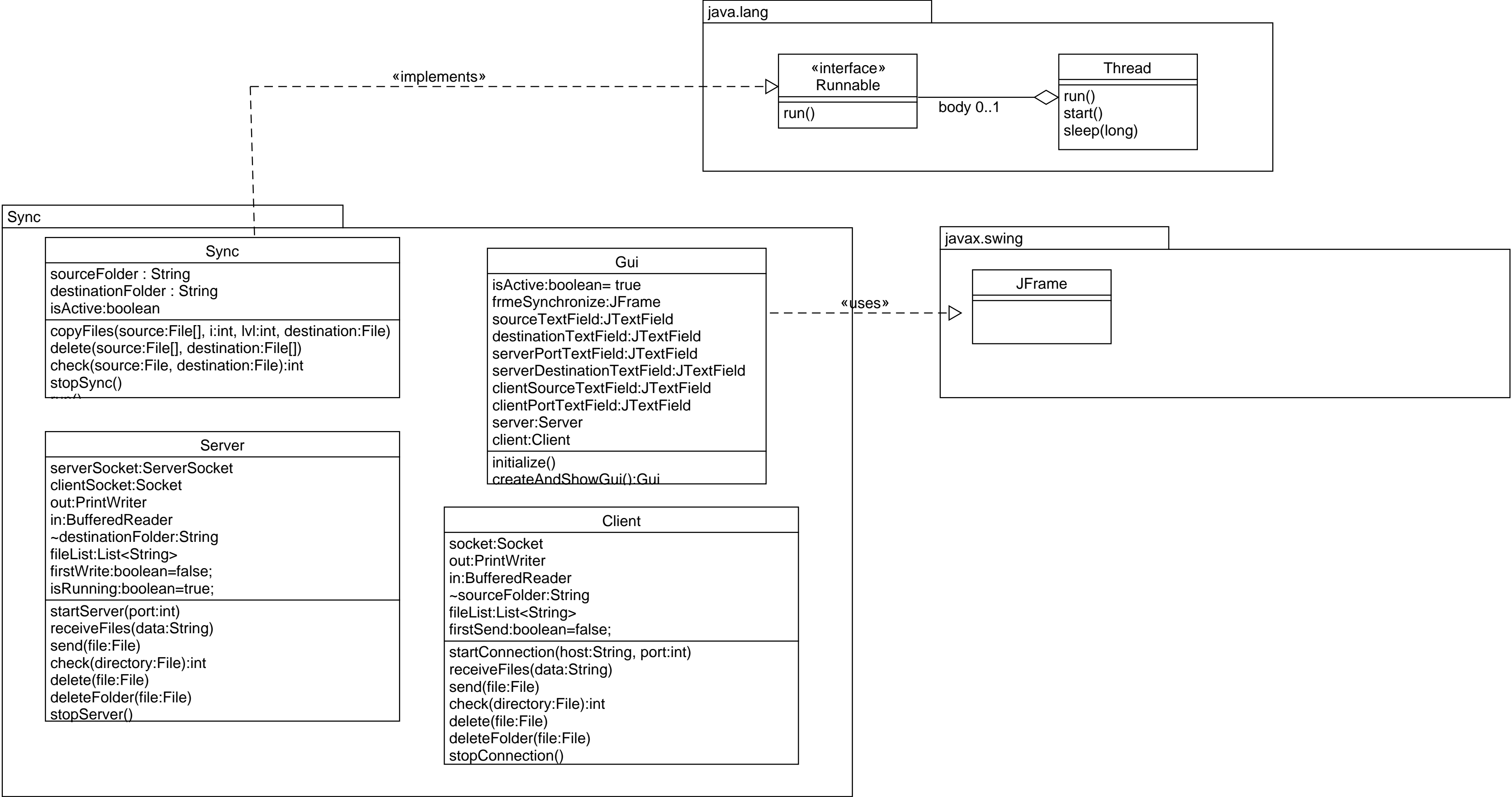
Server



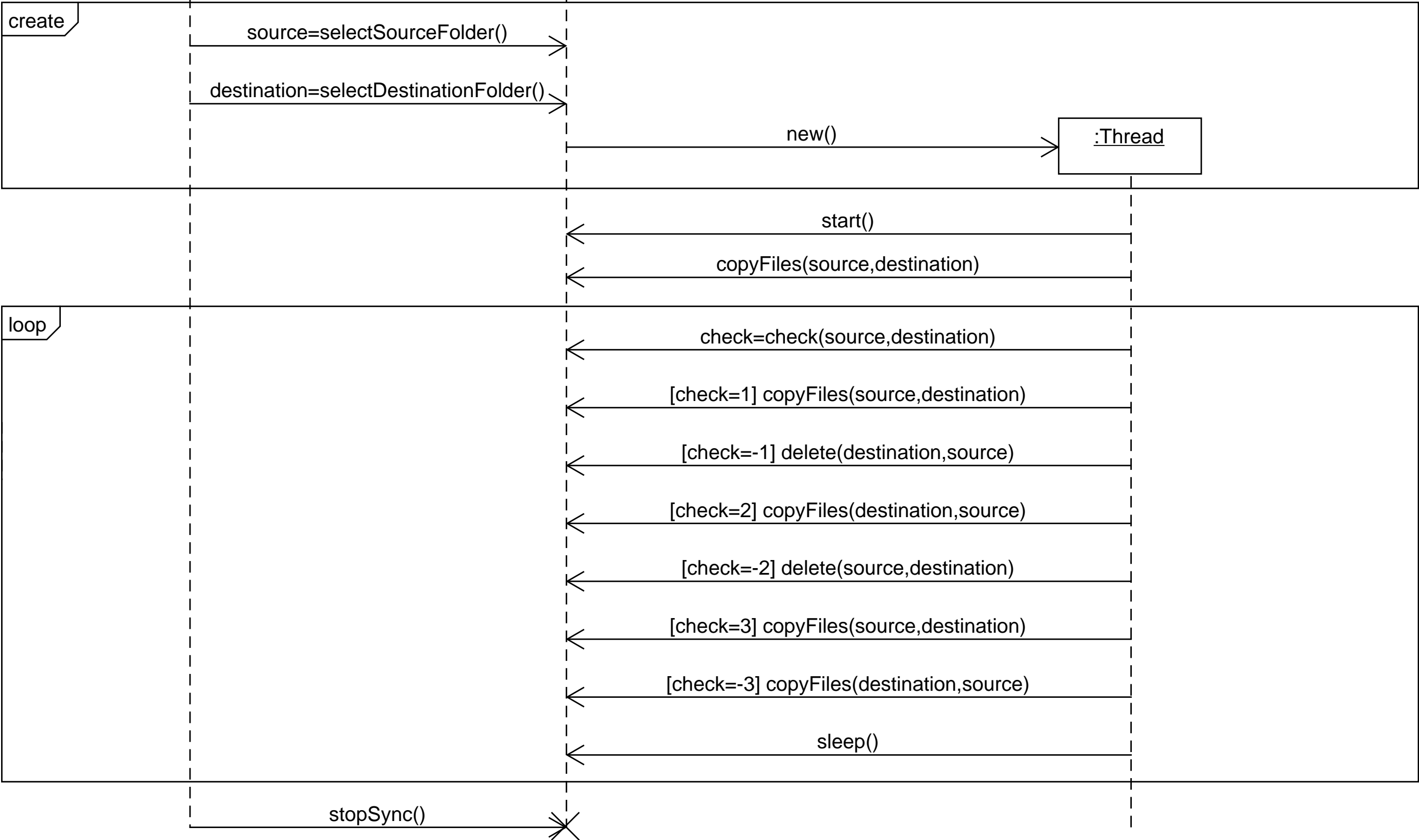
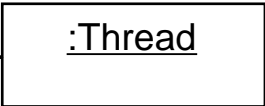
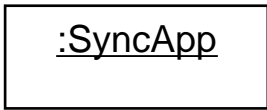
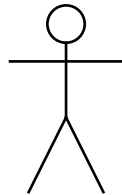
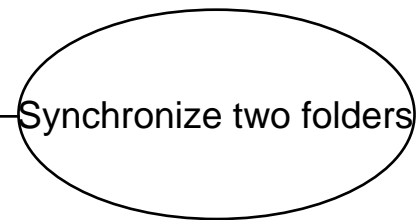
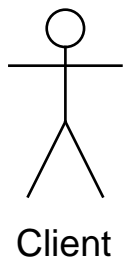
Class Diagram

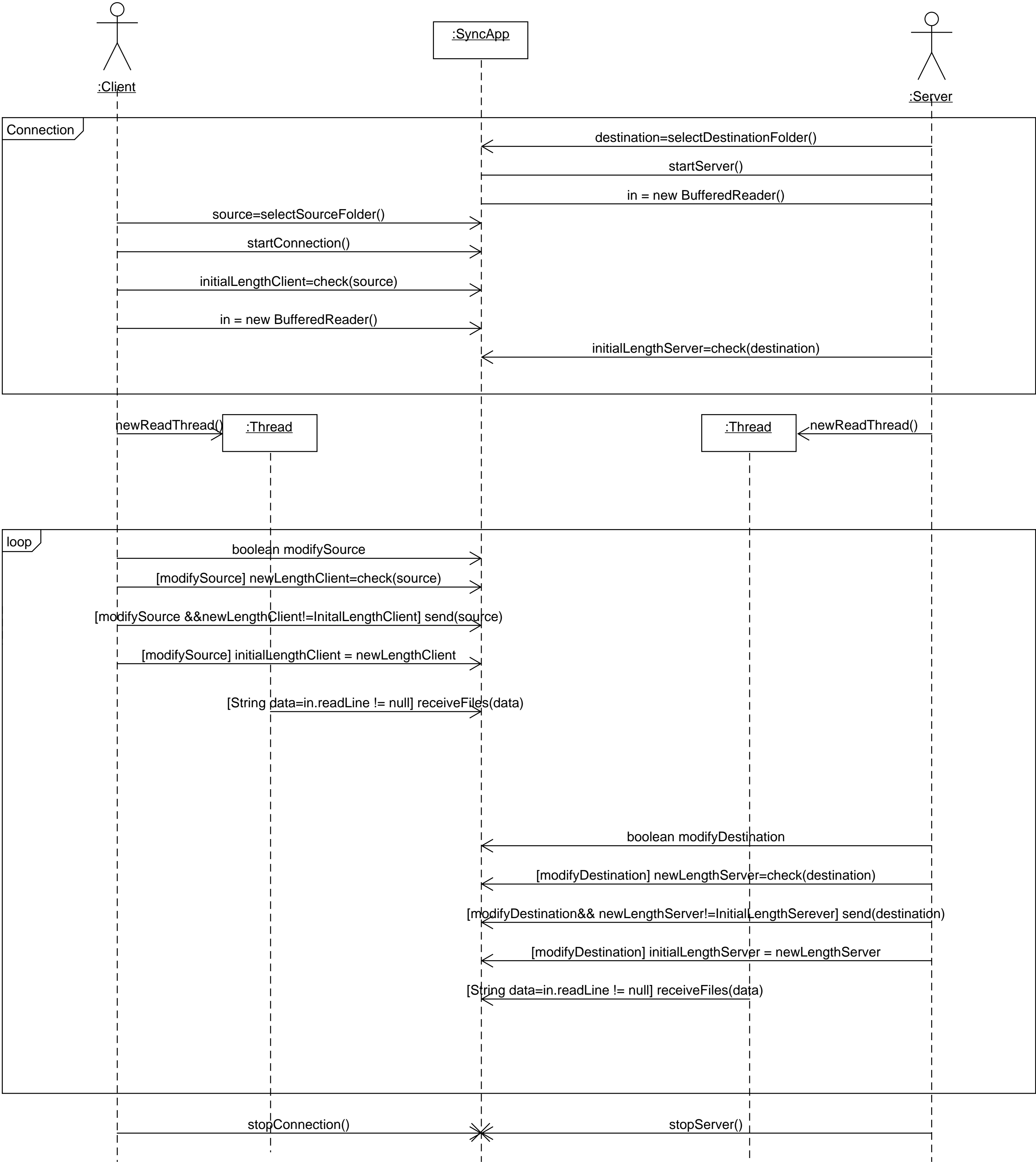
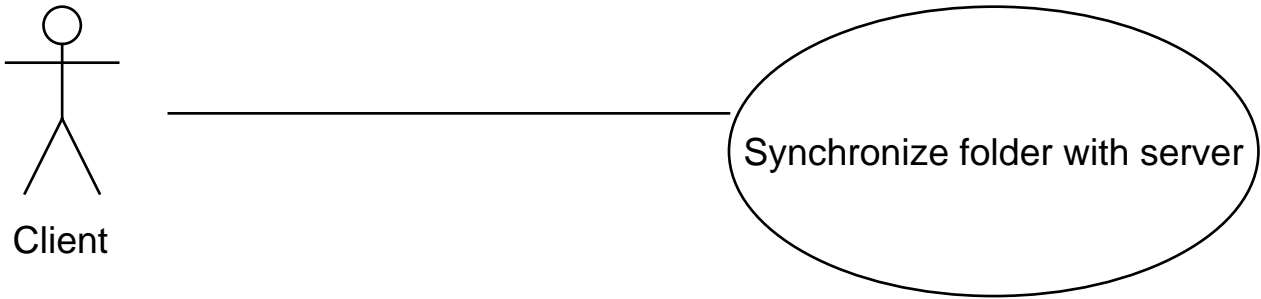


Package diagram

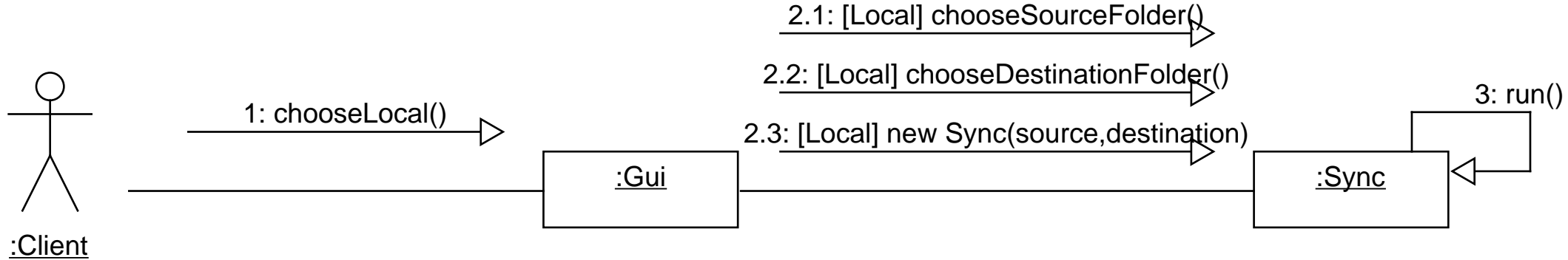


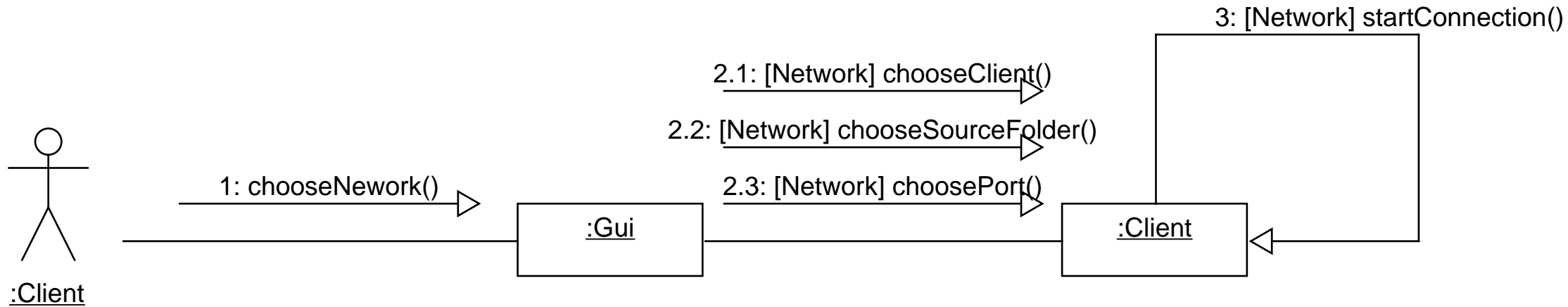
Sequence Diagram



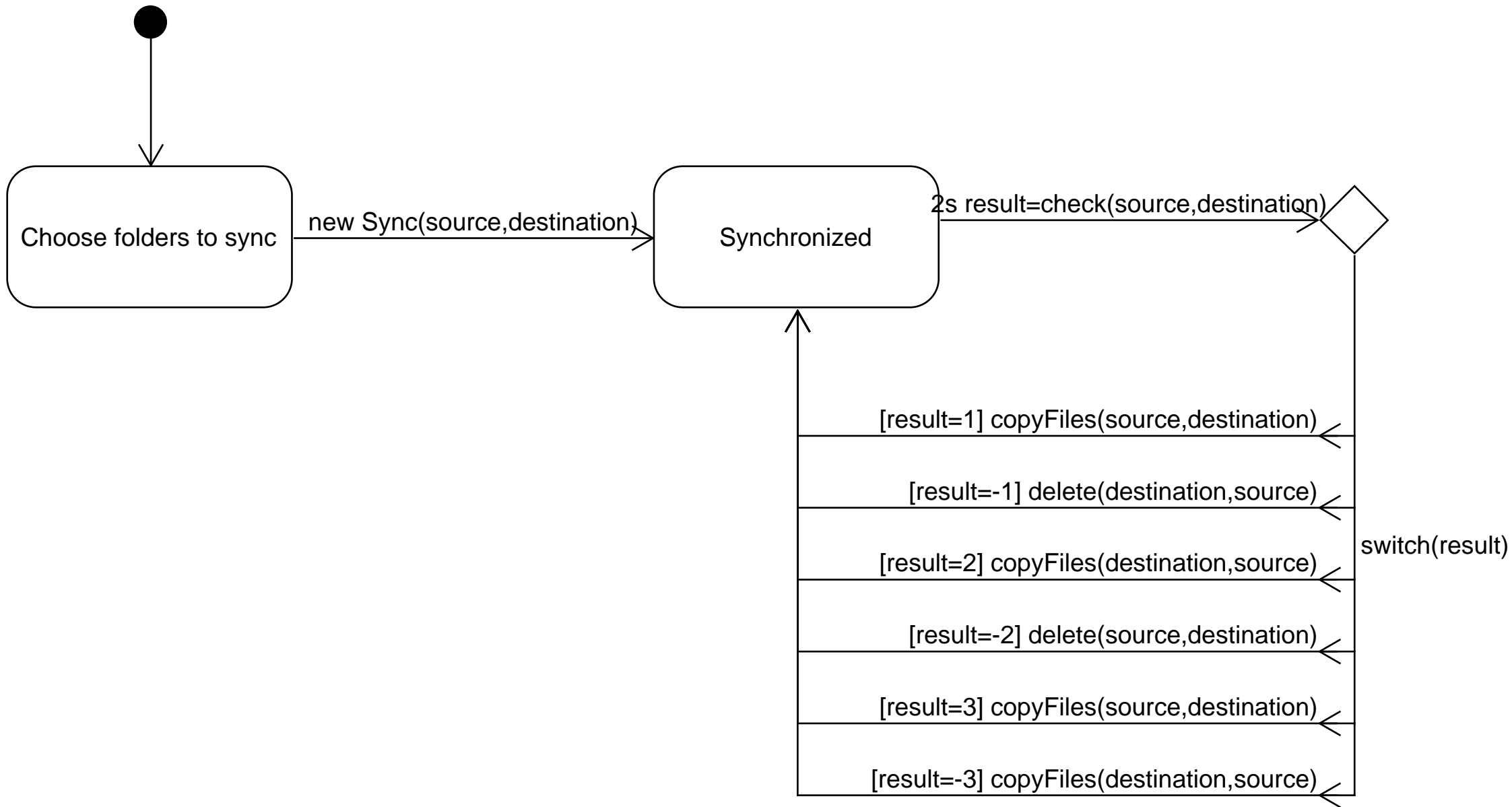


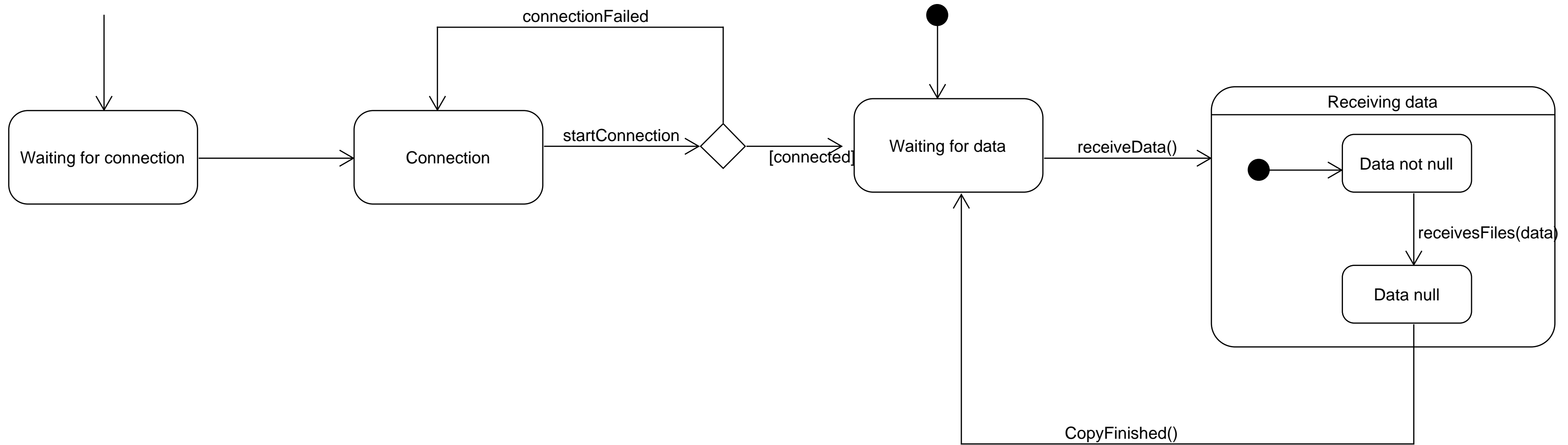
Communication diagram

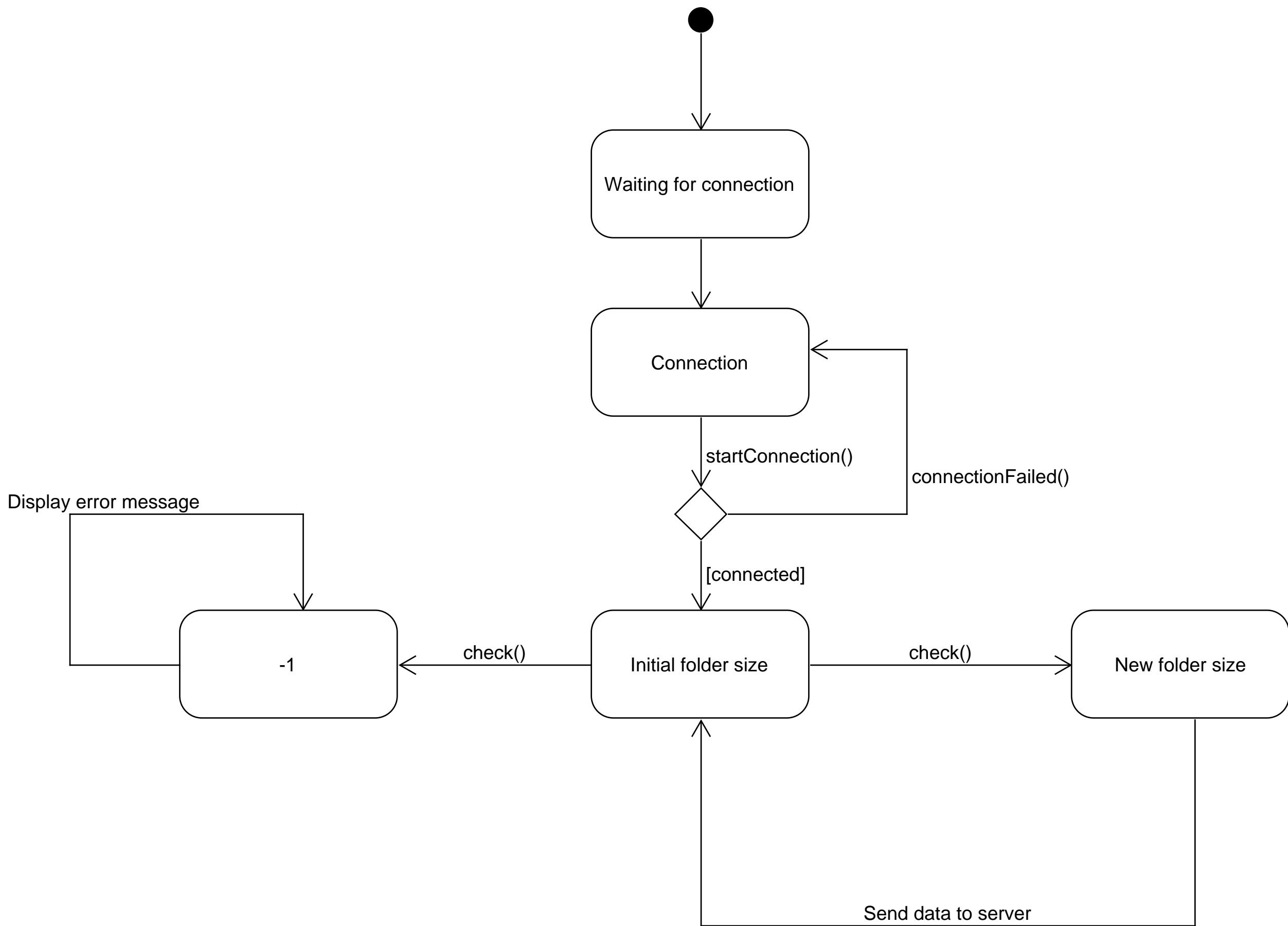




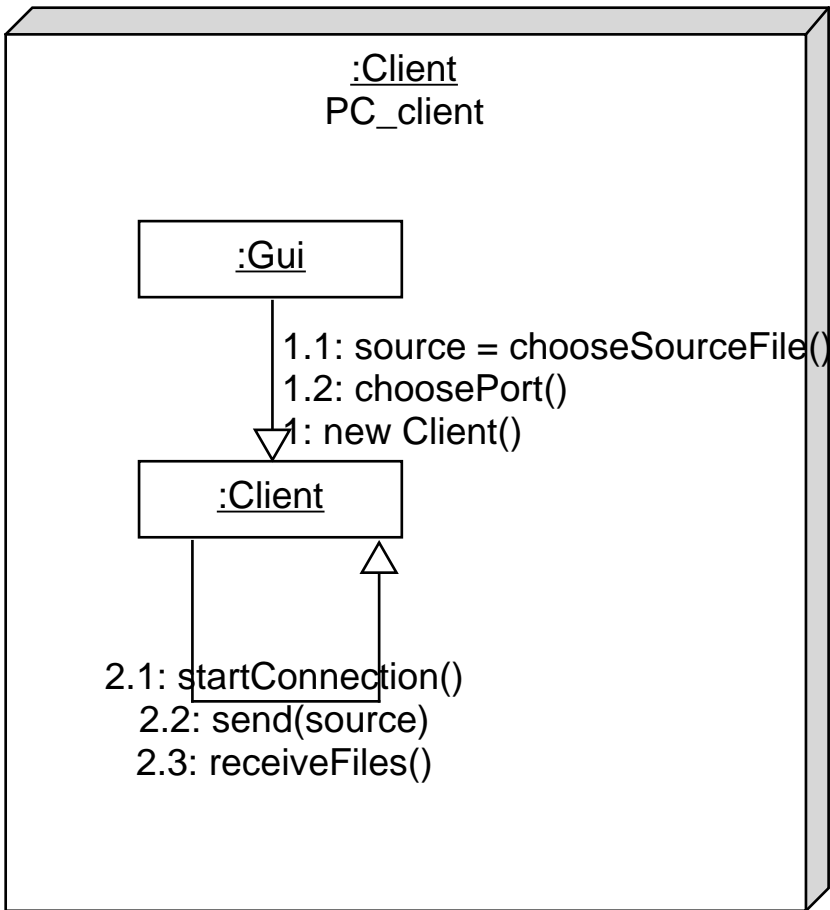
State diagram



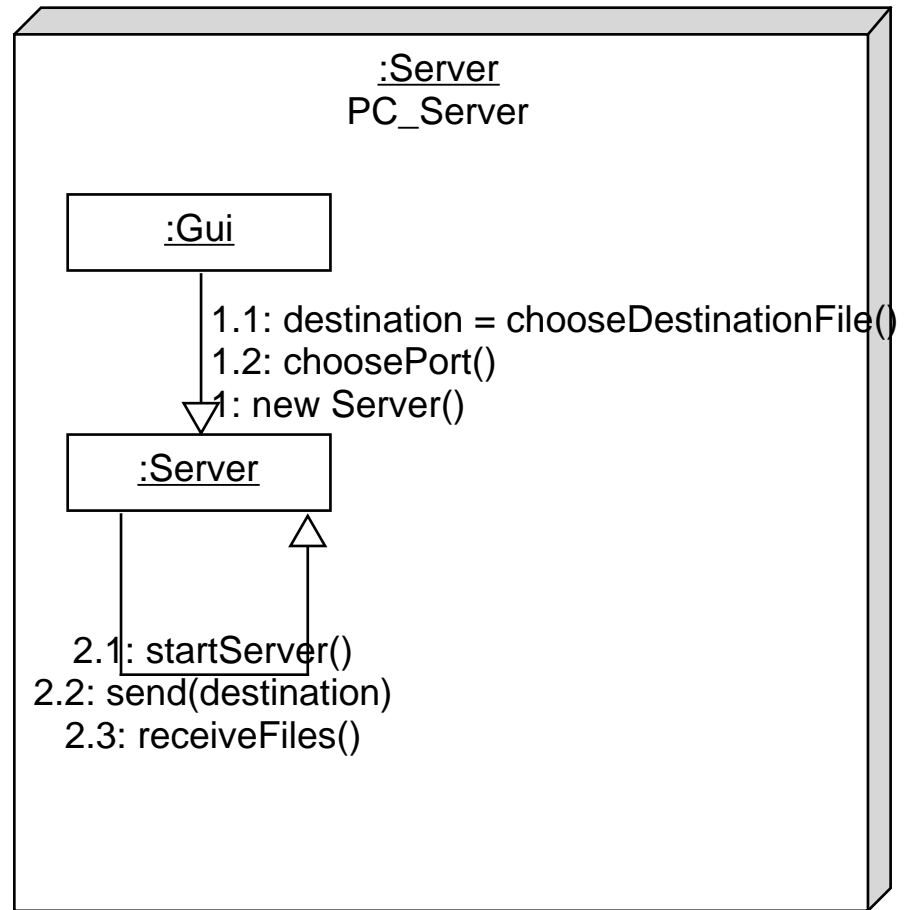




Deployment diagram



Client port
localhost



Server port
localhost

Local Sockets (TCP/IP)

Timing diagram

Network Synchronisation

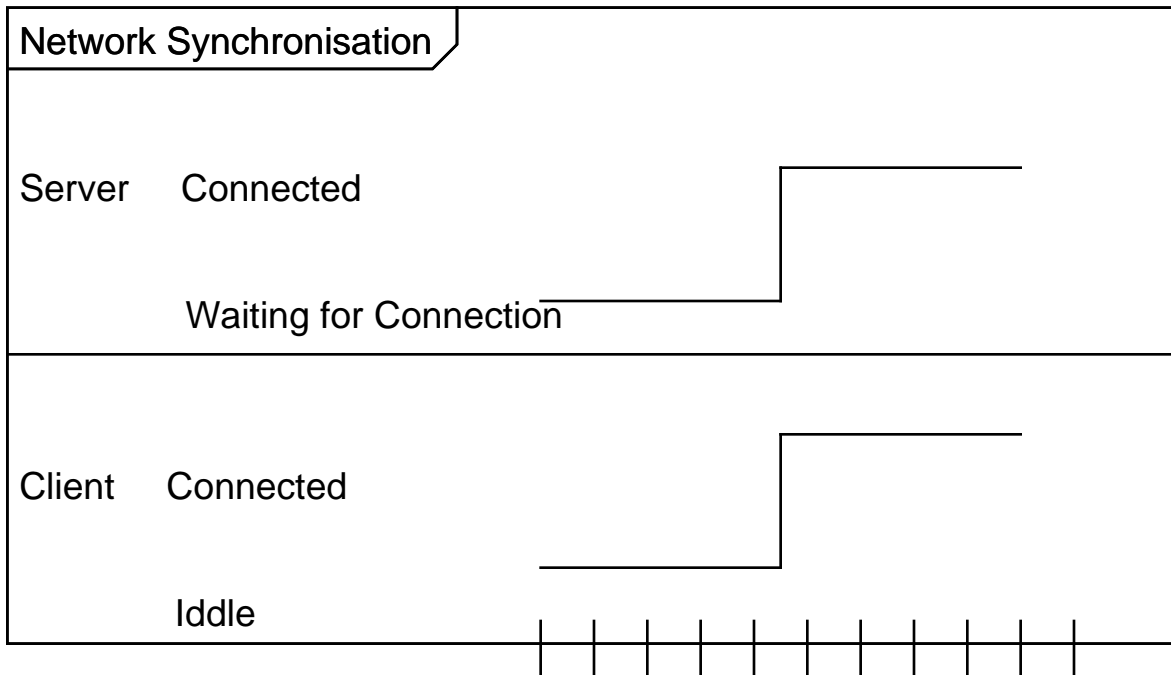
Server Connected

Waiting for Connection

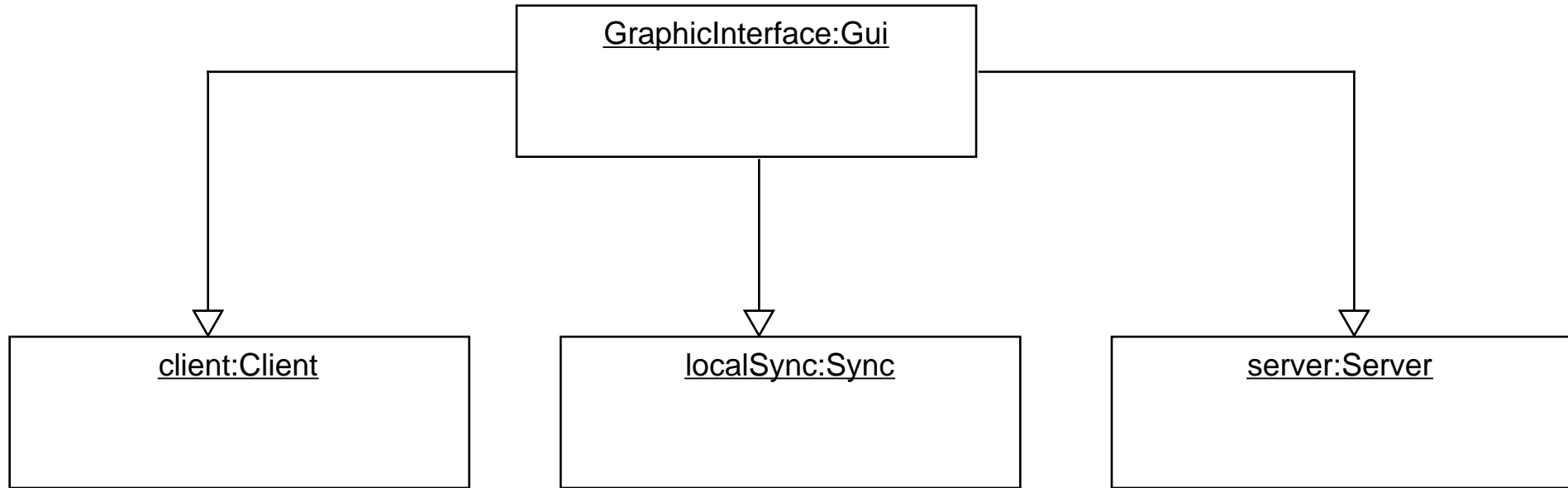
Client Connected

Idle

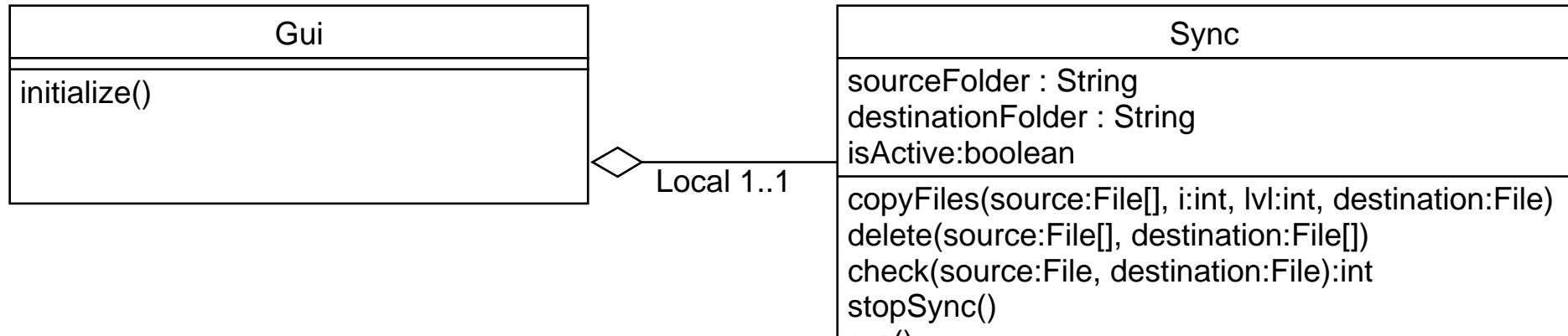
0s



Object diagram



OCL CONSTRAINTS

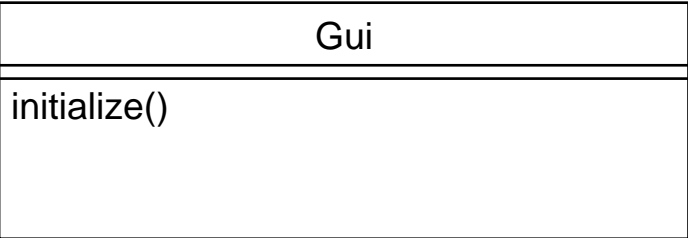


OCL

context sync.Sync

invariant valid_sourceFolder:

(sync.sourceFolder!=sync.destinationFolder)



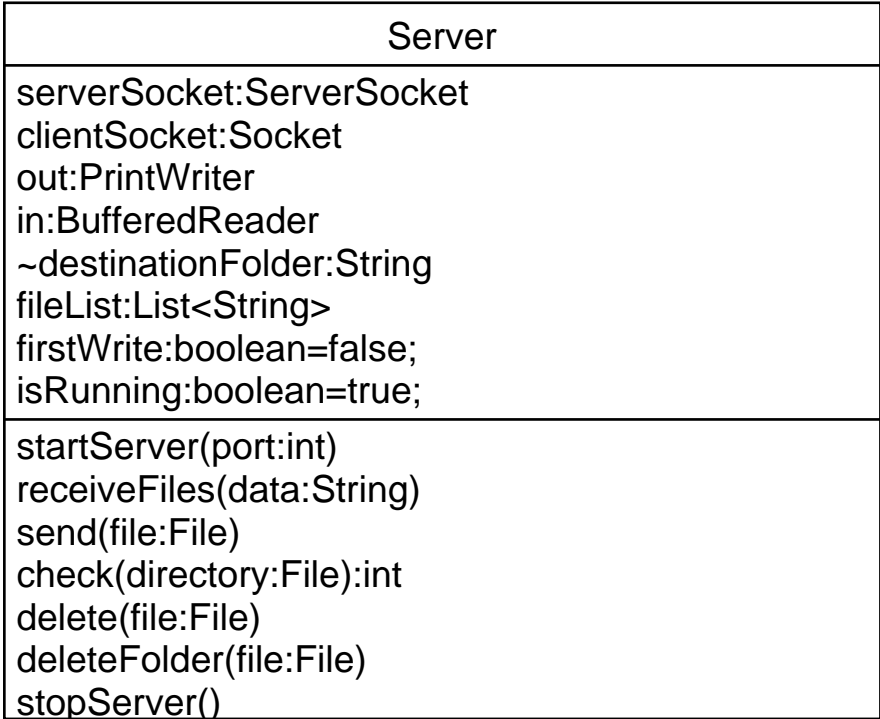
initialize()

1..1

Server

1..1

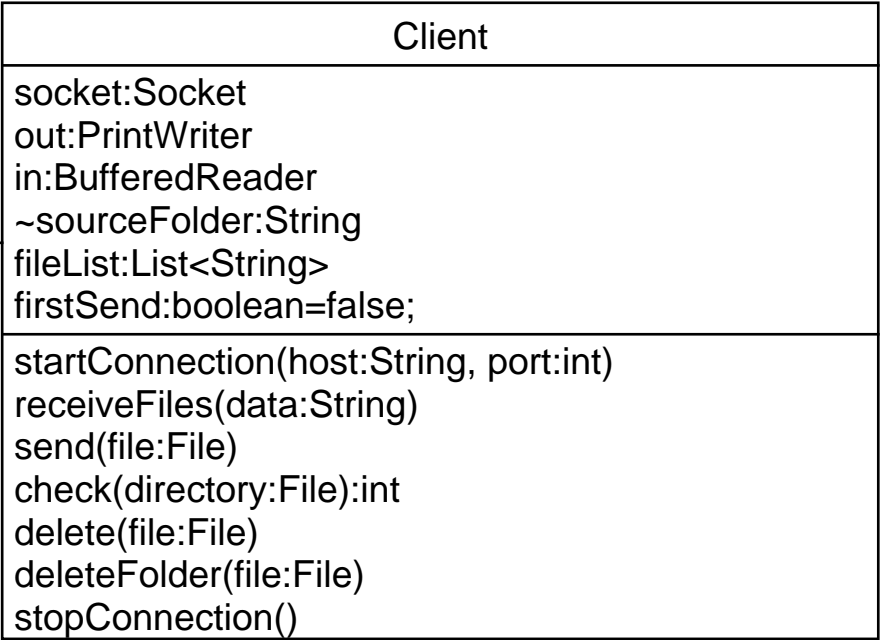
Client



Server

serverSocket:ServerSocket
clientSocket:Socket
out:PrintWriter
in:BufferedReader
~destinationFolder:String
fileList:List<String>
firstWrite:boolean=false;
isRunning:boolean=true;

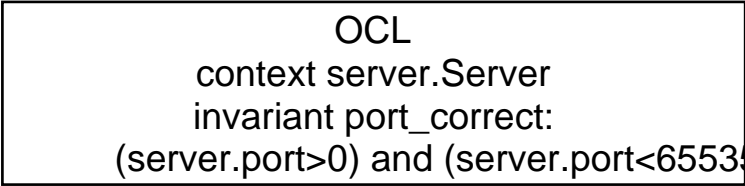
startServer(port:int)
receiveFiles(data:String)
send(file:File)
check(directory:File):int
delete(file:File)
deleteFolder(file:File)
stopServer()



Client

socket:Socket
out:PrintWriter
in:BufferedReader
~sourceFolder:String
fileList:List<String>
firstSend:boolean=false;

startConnection(host:String, port:int)
receiveFiles(data:String)
send(file:File)
check(directory:File):int
delete(file:File)
deleteFolder(file:File)
stopConnection()

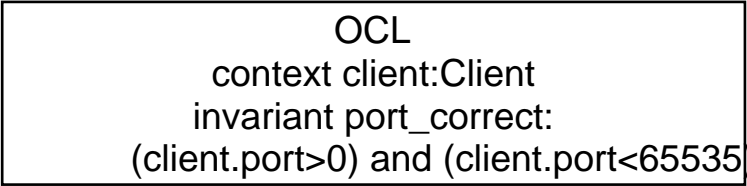


OCL

context server:Server

invariant port_correct:

(server.port>0) and (server.port<65535)

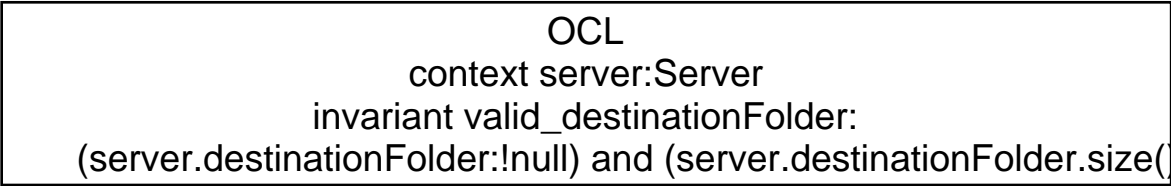


OCL

context client:Client

invariant port_correct:

(client.port>0) and (client.port<65535)

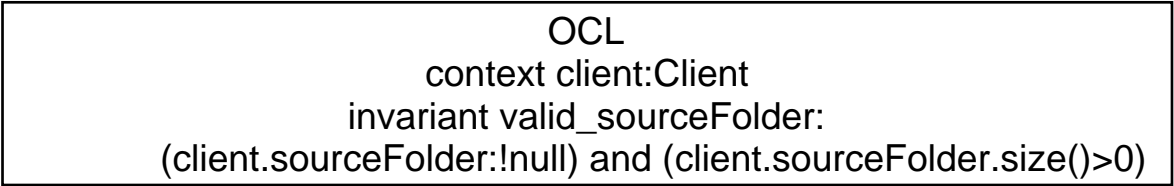


OCL

context server:Server

invariant valid_destinationFolder:

(server.destinationFolder!=null) and (server.destinationFolder.size()



OCL

context client:Client

invariant valid_sourceFolder:

(client.sourceFolder!=null) and (client.sourceFolder.size()